Einführungskurs

# Visual Basic 5.0

Christian Zahler

## 1 Einleitung

Windows-Programmieren – kompliziert und schwer zu lernen. Bisher eine Feststellung, gegen die auch eingefleischte Programmier-Fans keine Entgegnung fanden. Wochenlanges Arbeiten und intensives Beschäftigen mit technischen Details waren notwendig, um funktionsfähige Windows-Programme zu erstellen.

Mit der Einführung von Visual Basic auf dem Markt ist das nun einfacher geworden. Die Programmierung der Oberfläche wird nun "intuitiv" – so wie in Windows-Zeichenprogrammen durchgeführt. Erst dann benötigt man Kenntnisse der Programmierung; und diese müssen nicht extrem technisch sein.

### Geschichte

Visual Basic ist das "Enkelkind" der Programmiersprache BASIC (Beginners' All Purpose Symbolic Instruction Code = "Symbolische Anfänger-Programmiersprache für alle Zwecke"), die 1963 von John KERNENY und Thomas KURTZ entwikkelt wurde. BASIC war eine sehr einfache Sprache, mit der man sehr rasch Erfolge erzielen konnte. Leider war es nicht möglich, die Programme zu "gliedern". Ein Programm bestand daher anfangs aus einzelnen, numerierten Zeilen. In jeder dieser Zeilen fand sich ein Befehl, wie folgendes Beispiel zeigt:

```
10 REM BASIC-TESTPROGRAMM
20 PRINT "NAME: "
30 INPUT NAME$
40 PRINT "HALLO, ";NAME
50 END
```

Programme dieser Art werden sehr rasch unübersichtlich; in Programmiererkreisen nennt man noch heute derartige Programme "Spaghetti-Programme", da Meter für Meter derartige Befehlsfolgen ausgeworfen wurden. Lange Zeit galt BASIC daher als "Spielzeug-Programmiersprache", die von all jenen verwendet wurde, die nicht ernsthaft die Absicht hatten, programmieren zu lernen.

Mitte der 70er Jahre entwickelte Microsoft eine BASIC-Version, die mit MS-DOS mitgeliefert wurde und als GWBASIC bekannt wurde. Da aber die Programme zu langsam waren und außerdem nicht eigenständig abgearbeitet werden konnten, traf für diesen "Dialekt" von BASIC der Vorwurf des Spielzeugs durchaus zu. 1982 trat QBASIC die Nachfolge von GWBASIC an: Hier wurden die Zeilennummern weggelassen, auch die Möglichkeit gegeben, eigenständig ausführbare Programme zu erstellen.

Seit Sommer 1991 gibt es Visual Basic, eine Programmierumgebung für Windows. Mit dieser wollen wir uns in der Folge beschäftigen; die aktuelle Version 5.0 erschien Anfang 1997. In diesen Jahren hat es sich von einer "Spielzeugsprache" zu einem ernstzunehmenden Werkzeug zur Entwicklung von **Datenbank-Anwendungen** und Client/Server-Anwendungen gewandelt.

**Wichtig:** Visual Basic 5.0 ist ein reines 32 bit-Entwicklungssystem, laeuft also nur auf Windows 95 (98) und Windows NT..

VB 5.0 enthält **Visual Basic for Applications (VBA) 5.0**, das erstmals in Anwendungen wie Excel 5.0 oder Project 4.0 enthalten war.

Microsoft plant, VB zur generellen "Entwicklungssprache" innerhalb der Windows-Welt zu machen.

Derzeit wird Visual Basic bereits in folgenden Microsoft-Produkten unterstützt:

- Office 97 (= Word 97, Access 97, Excel 97): enthält VBA 5.0
- Project 98
- Internet Explorer 3.x und 4x
- Exchange Server 5.0

In den nächsten Jahren soll VB auch in folgenden Microsoft-Produkten enthalten sein:

- Windows NT 5.0
- SQL-Server

Die weitere Entwicklung von Visual Basic wird auch von den Konkurrenzprodukten abhängen, mit denen sich ebenfalls leistungsfähige Windows-Programme erstellen lassen. Dazu zählen etwa:

- C++ (Microsoft Visual C++, Borland C++, Symantec C++)
- Delphi (Borland)
- Microsoft Visual J++ (Weiterentwicklung von Visual C++ Richtung Internet)

### Versionen von Visual Basic 5.0

Visual Basic 5.0 wird in drei Versionen ausgeliefert:

- **Standard-Ausgabe:** dient zur Entwicklung von 32 bit-Programmen in Windows 95 oder Windows NT. Fast alle Erklärungen in diesem Skriptum beziehen sich auf die Standard-Ausgabe. Als Betriebssystem wurde Windows 95 verwendet.
- **Professionelle Ausgabe:** enthällt viele zusätzliche Steuerelemente (etwa Internet-Steuerelemente) und Möglichkeiten, eigene Steuerelemente zu entwickeln.
- **Enterprise-Ausgabe:** für die Erstellung größerer Programmierprojekte in Teamarbeit, für Client/Server-Anwendungen, enthält Werkzeuge zur Datenbankverwaltung, ein Versionsverwaltungssystem (MS-Visual Source Safe) und vieles mehr.

### Neues in der Version 5

- **System Code:** Visual Basic 5 enthält einen Compiler, der deutlich schnelleren Systemcode erzeugt. (N*ur in Professional und Enterprise Edition*.)
- Unterstützung von **ActiveX-Steuerelementen**. ActiveX-Steuerelemente sind Objekte (etwa Schaltflächen), die Webfähigkeiten aufweisen.
- Die Erzeugung von Programmen mit einer Explorer-ähnlichen Oberfläche ist möglich.
- Die Erzeugung von **voll internetfähigen Programmen** ist möglich!
- OLE Drag & Drop: Die meisten Steuerelemente unterstützen Drag & Drop, etwa kann der Inhalt eines Word-Dokuments in ein Textfeld gezogen werden.
- Erstellung eigener ActiveX-Steuerelemente (N*ur in Professional und Enterprise Edition*!)

### Vor- und Nachteile von Visual Basic

#### Vorteile

- Visual Basic ist eine strategische Sprache von Microsoft, hat daher Zukunft.
- Visual Basic ist modular aufgebaut und durch Zusatzsteuerelemente erweiterbar.
- Visual Basic ist leicht erlernbar.
- Visual Basic hat eine hervorragende Datenbank-Schnittstelle.

#### Nachteile

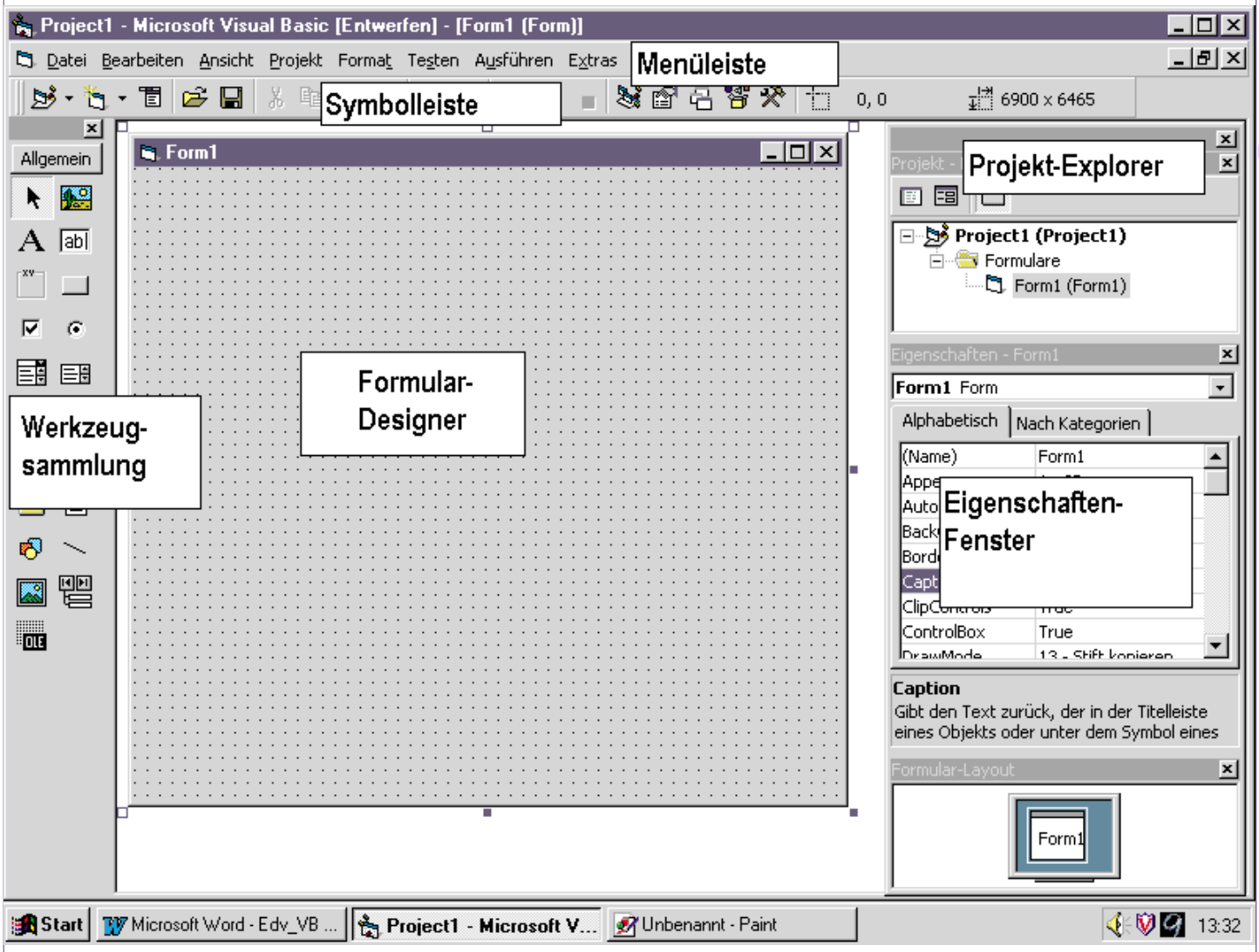
- In VB gibt es keine Pointer.
- VB ist nicht sehr hardwarenahe; der direkte Zugriff auf serielle Schnittstellen oder Hardwarekomponenten muß über DLL-Funktionen erledigt werden.
- VB ist nicht vollständig objektorientiert.
- VB-Programme sind langsam.


christian.zahler@telecom.at

# 2 Grundlagen der Programmierung mit Visual Basic

## 2.1 Start

Beim Start erscheint folgender Bildschirm:

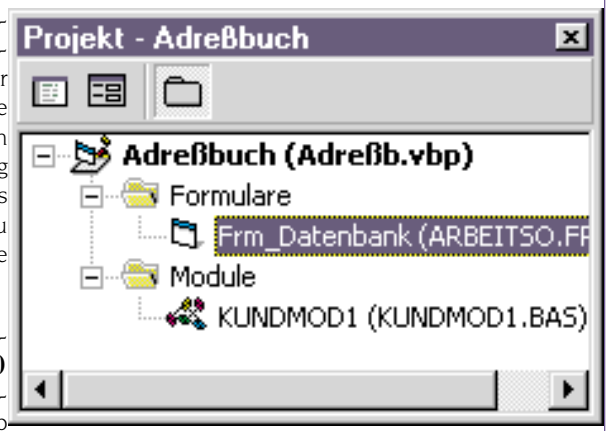



Bemerkung: Die Anordnung der einzelnen Fenster kann von Fall zu Fall etwas anders aussehen.

### Wie arbeitet Windows?

Windows ist ein Betriebssystem, das die zentrale Aufgabe hat, sogenannte **Fenster** zu verwalten. Ein Fenster ist einfach ein rechteckiger Bereich mit eigener Begrenzung. Beispiele für Fenster: Dialogfeld "Öffnen", Programmfenster "Paint", Fehlermeldungsfenster. Auch Symbole, Befehlsschaltflächen (etwa "Ok") oder Menüleisten sind spezielle Arten von Fenstern.

Windows verwaltet alle diese Fenster, indem jedem Fenster eine eindeutige Nummer zugeordnet wird (die **Fenster-Zugriffsnummer**). Das System überwacht jedes Fenster laufend auf Anzeichen von **Ereignissen** (etwa: wird darauf geklickt?). Wenn ein derartiges Ereignis (Klicken, Tastendruck) eintritt, so wird eine sogenannte **Meldung** an das Betriebssystem ausgelöst, die vom System verarbeitet und an die anderen Fenster gesendet wird. Jedes Fenster kann dann selbständig auf diese Meldungen reagieren (etwa: sich neu zeichnen). Die Entwicklung eines Visual Basic-Programms beruht also darauf, **Fenster** zu definieren und auf bestimmte **Ereignisse** zu reagieren.




Zunächst muß man eine sogenannte **"Form" (Formular)** entwickeln. Unter einer Form versteht man das Fenster, innerhalb dessen das Programm später ausgeführt werden soll.

Alle Objekte können im Fenster "Eigenschaften" verändert werden.

Ein **Visual Basic-Projekt** besteht stets aus mehreren Dateien:

- einer **Projektdatei** mit der Extension `*.VBP` (bis Version 3: `*.MAK`), welche alle Komponenten des Projekts definiert. In dieser Datei ist eine Übersicht aller Dateien enthalten, die zum Projekt gehören; während des Betriebs läßt sich diese Datei auch mittels des Projekt-Explorers (Standardname: `Projekt1`) ansehen.
- für jede **Form** eine Datei mit der Extension `*.FRM` (zu Beginn nur eine Datei, weil nur eine Form erstellt wurde); diese Datei enthält nicht nur die einzelnen Objekte, son-

dern auch den zugehörigen Programmcode.

- eine **Binärdatendatei** mit der Extension `*.FRX` für alle jene Formulare, die Steuerelemente mit binären Eigenschaften enthalten (etwa Picture-Eigenschaft). Diese Dateien können nicht bearbeitet werden.
- Programmcode, der nichts mit Objekten in einer Form zu tun hat, findet sich mit **"Modulen"** mit der Endung `*.BAS`
- Die Eigenschaften und Methoden eines Objekts können in sogenannten **Klassendateien** mit der Endung `*.CLS` definiert werden.
- **ActiveX-Steuerelemente** haben die Erweiterung `*.OCX`. (Anmerkung: Das Konzept der `*.VBX`-Dateien wurde aufgelassen; es wurde in den 16 bit-Versionen unterstützt.) Diese Steuerelemente sind OLE-fähig: Es ist denkbar, daß man beispielsweise Excel als "Taschenrechner" mißbraucht und die Ergebnisse im eigenen Programm weiterverwendet. Diese Steuerelemente sind in der Lage, auf einer Webseite mit dem Internet zu kommunizieren.
- **Ressourcendateien** mit der Erweiterung `*.RES` enthalten Bitmaps und Strings, die etwa zur Anpassung an verschiedene Landessprachen benötigt werden.

### Was sind Steuerelemente (englisch "Controls")?

Steuerelemente sind Objekte, die auf einer Form plaziert werden und Eingaben gestatten. Bekanntestes Beispiel ist eine **Befehlsschaltfläche**, die in Windows 95 etwa folgendermaßen aussehen kann:


Eine Befehlsschaltfläche kann angeklickt werden, worauf bestimmte Aktionen ausgeführt werden.

### Visual Basic im WWW

Die Microsoft-WWW-Site zu Visual Basic enthält verschiedene Bereiche, die für Visual Basic-Programmierer von Interesse sein könnten.

Die Visual-Basic-Homepage finden Sie unter der Adresse:

*http://www.eu.microsoft.com/germany/produkte/vbasic/* oder

*http://www.microsoft.com/germany/produkte/vbasic/* (langsamerer Zugriff)

Hier finden Sie unter anderem:

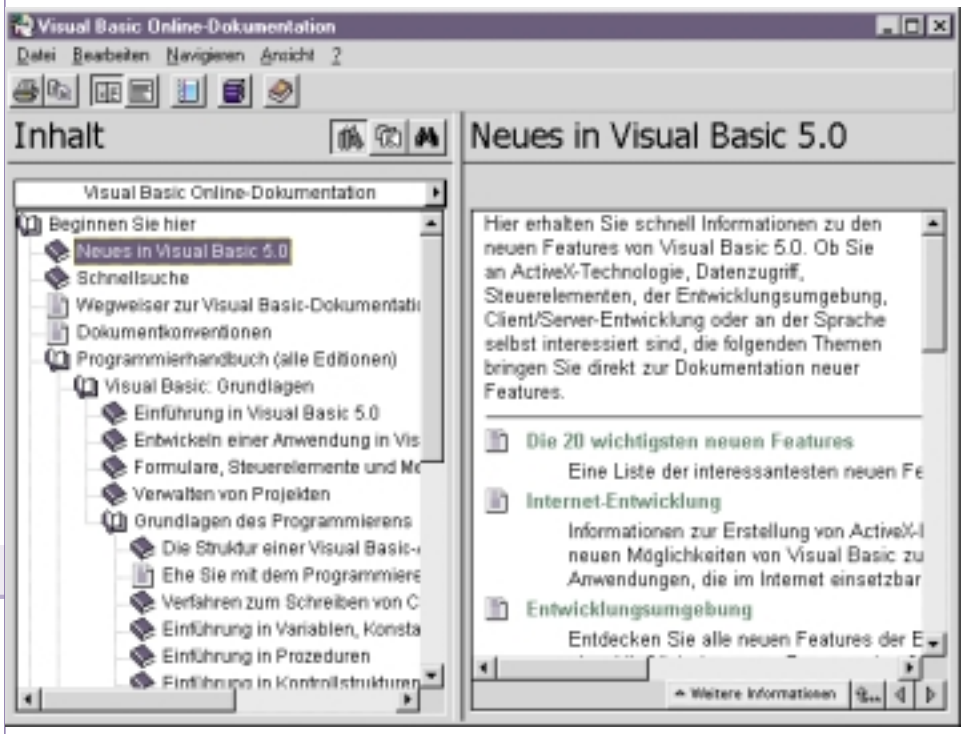
✓ Die MS-Visual Basic-Knowledge Base, eine Sammlung von Anleitungen, Fehlerkorrekturen und Problemlösungen.

✓ Die Visual Basic Software Library mit Updates zu Programm- und Hilfedateien, Gerätetreibern usw.

✓ Visual Basic Fragen und Antworten enthält Antworten auf häufig gestellte Fragen an das MS Software Service.

### Online-Dokumentation

Rufen Sie die Online-Dokumentation mit [***Start***]-[***Programme***]-[***Visual Basic 5.0***]-[***Online-Dokumentation***] auf. Diese enthält die gesamte Visual Basic-Dokumentation gesammelt in einzelnen elektronischen "Büchern".



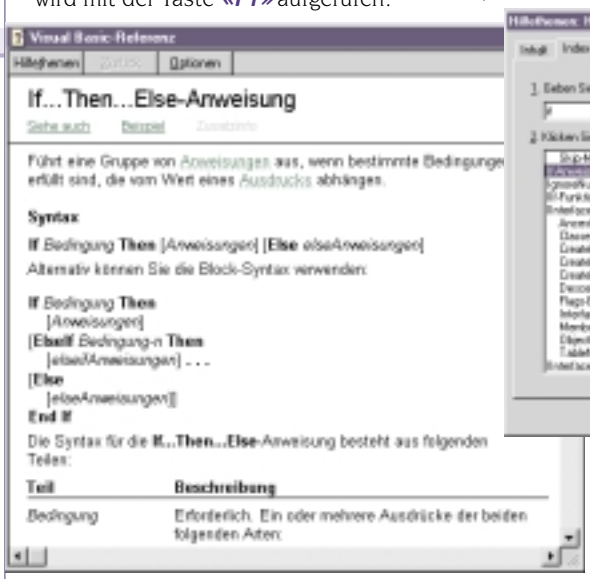

Klickt man das unterstrichene Wort "***Beispiel***" an, so findet man ein lauffähiges Codebeispiel, welches durch Bearbeiten – Kopieren sofort im aktuellen Projekt ausprobiert werden kann.

Selbstverständlich kann auch themenmäßig gesucht werden. Hier bietet sich der Menüeintrag Hilfe – Suchen an:

**Wichtig:** Eine Programmiersprache wie Visual Basic kann **nur** bei konsequenter Benützung der Online-Hilfe wirklich erlernt werden können!

### Online-Hilfe

Unübertroffen in Visual Basic ist die Online-Hilfe. Sie bietet ausführliche Hilfestellungen mit Beispielen zu allen Sprachelementen – ob es sich um Befehle, Steuerelemente oder Eigenschaften handelt. Sie wird mit der Taste ***«F1»*** aufgerufen.



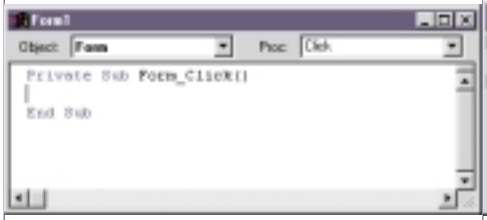

Hat man eine Anweisung markiert, so erscheint bei Drücken der ***«F1»***-Taste sofort die Hilfe zum markierten Wort. Beispiel: Hilfe zur `if`-Anweisung.

Beispiel 1: "Hallo" (B01)

Als traditionell erstes Programm wollen wir ein "Hallo"-Programm schreiben, welches nichts anderes tut, als uns zu begrüßen.

Dazu wählt man zunächst im Menü ***Datei*** den Eintrag ***Neues Projekt***.

Dann klicken wir doppelt auf das "***Form***"-Fenster; es öffnet sich ein Code-Fenster, in das die Visual Basic-Befehle geschrieben werden:



Wir wählen bei "***Objekt***" den Eintrag `Form`, bei "***Proc***" (Prozedur) den Eintrag "`Click`". Das bedeutet, daß alles, was wir jetzt schreiben, durch einfaches Klicken mit der Maus auf die Form gestartet wird.

## 2.2 Einige Bemerkungen zur Programmierung

1 Alle Visual Basic-Programme bestehen aus abgetrennten Programmteilen = Unterprogrammen. Man unterscheidet **Prozeduren** (`Sub`) und **Funktionen** (`Function`).

2 **Ereignisgesteuerte Programmierung:** Jede Prozedur beginnt mit dem Wort `Sub` und endet mit `End Sub`. Nach dem Wort `Sub` steht der Name der Prozedur. Dieser Name ist im allgemeinen nicht frei wählbar, sondern besteht aus dem Namen des Objekts und dem Namen der Aktion (des **Ereignisses**), bei der das Unterprogramm ausgelöst werden soll.

Jede Prozedur besteht aus Zeilen, in jeder Zeile steht eine Visual Basic-Anweisung.

**Gültigkeitsbereich von Prozeduren:** Hier unterscheidet man drei Zugangsstufen:

```
Private Sub MeineProzedur()
```

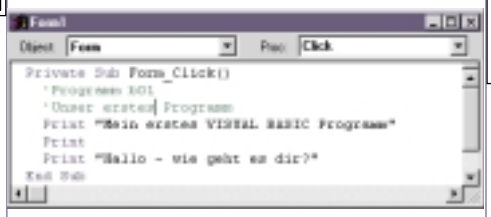
Diese Prozeduren können innerhalb der gleichen Form (des gleichen Moduls, der gleichen Klasse) verwendet werden.

```
Public Sub MeineProzedur()
```

Diese Prozeduren können in der gesamten Applikation verwendet werden, indem man den Namen der Form voranstellt, in diese Prozedur gespeichert ist, etwa

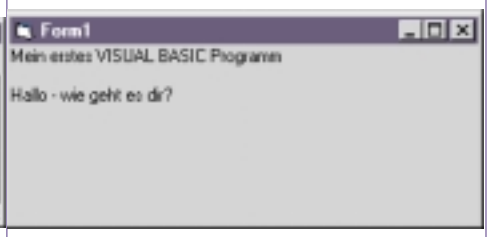
```
Form1.MeineProzedur
```

Dann schreiben wir folgendes:



Wenn wir das Programm ausführen wollen, wählen wir ***Ausführen – Start*** oder drükken die Funktionstaste ***«F5»***.

Als Ergebnis erhalten wir:




### Abspeichern

Pulldown-Menü ***Datei – Projekt speichern***.

Als Standard-Dateinamen wird für Formen `Form1.frm`, für Projekte `Projekt1.vbp` vorgeschlagen.

### *Wichtige Hinweise*

- Es empfiehlt sich, für jedes Projekt ein **eigenes Unterverzeichnis** (einen eigenen Ordner) anzulegen.
- Unbedingt sollte man **nicht den Standardnamen verwenden** (dies gilt sowohl für Projekt- als auch für Formdateien), sondern eigene Namen vergeben. Grund: Falls keine eigenen Unterverzeichnisse angelegt worden sind, können sehr leicht Dateien überschrieben werden!

### Debugging (Fehlerbehebung)

Wesentlich ist es, bei Auftreten von Fehlern Programme im Einzelschrittmodus (also Zeile für Zeile) durchlaufen zu können. Dies geschieht mit der Funktionstaste ***«F8»***. Mit der ***«F9»***-Taste können Haltepunkte gesetzt werden, damit das Programm beim Eintreffen bestimmter Ereignisse anhält.

## 2.3 Variablen

**Beispiel 2: Variablen, Ein- und Ausgaben (B02)**

Alle Daten, die in einem Programm verwendet werden sollen, werden in "Variablen" gespeichert. "Variable" deshalb, weil der Inhalt veränderlich ist.

Variablen müssen benannt werden; der Variablenname darf max. 40 Zeichen umfassen, dabei dürfen ***keine Leerzeichen und keine Sonderzeichen*** verwendet werden. Groß- und Kleinschreibung werden als gleichbedeutend behandelt.

Es ist vorteilhaft, Variablen vor ihrer Verwendung zu ***deklarieren***. Dies geschieht mit der Anweisung `Dim`.

Mit dem Menüeintrag ***Extras – Optionen – Umgebung – "Variablendeklaration erforderlich"*** (in der englischen Version Menüpunkt ***Tools***!) legt man fest, daß Variablen vor ihrer Verwendung deklariert werden **müssen**.

Es muß allerdings auch der Datentyp (Variablentyp) angegeben werden, das heißt, es muß festgelegt werden, ob man ganze Zahlen, Dezimalzahlen oder Zeichenfolgen ("Texte") in einer Variablen abspeichern will.

Dazu bestehen zwei Möglichkeiten:

1 ""Typkennzeichen" Je nach Datentyp wird am Ende des Variablennamens ein "Typenkennzeichen" angehängt; bei Zeichenfolgen etwa ein $-Zeichen, bei ganzen Zahlen ein %-Zeichen.

Beispiel: `Dim Name$, Monat%`

2 Der Datentyp wird in der Deklaration angegeben:

Beispiel:

```
Dim Name As String, Monat As Integer
```

Hängt man **kein** Typkennzeichen an bzw. gibt keinen Datentyp an, so liegt der Variablentyp **"Variant"** vor, der "sinngemäß" auf die eingegebenen Daten reagiert.

Einer Variablen kann sehr einfach ein Wert **zugewiesen** werden; der Operator "=" dient dazu. Er bewirkt, daß der Ausdruck rechts vom "="-Zeichen berechnet und in den Speicherplatz der Variablen geschrieben wird.

Beispiele:

```
Name$ = "Peter"
Monat% = 10
S = K * (1 + P / 100)
I = I + 1
```

Die letzte Zeile verdient besondere Aufmerksamkeit. Man sieht, daß die Verwendung des Zuweisungsoperators "=" **nicht**

| Datentyp | Typenkennzeichen | Speichergröße (in Byte) | Bereich |
|---|---|---|---|
| **Boolean** | | 2 | 0 (False) oder -1 (True) |
| **Byte** (Ganzzahl) | | 1 | 0 bis 255 |
| **Integer** (Ganzzahl) | % | 2 | -32.768 bis 32.767 |
| **Long** (lange Ganzzahl) | & | 4 | -2.147.483.648 bis 2.147.483.647 |
| **Single** (Gleitkommazahl mit einfacher Genauigkeit) | ! | 4 | -3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte; und 0. |
| **Double** (Gleitkommazahl mit doppelter Genauigkeit) | # | 8 | -1,79769313486232E308 bis -4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte; und 0. |
| **Date** | | 2 | 1. Januar 100 bis 31. Dezember 9999 |

dem mathematischen Gleichheitszeichen entspricht. Der Zuweisungsoperator holt in diesem Fall den Wert der Variablen I aus dem Speicher (dieser Wert sei zum Beispiel 5), erhöht ihn um 1 (ergibt 6) und schreibt den neuen Wert wieder an die Speicheradresse von I (damit ist der Wert 5 durch den neuen Wert 6 überschrieben worden).

### Geltungsbereich von Variablen

**a) In Modulen**

- `Public A` gilt in allen Prozeduren aller Module und Forms, die zum Projekt gehören **(globale Variable)**
- in den Allgemeinen Deklarationen des Moduls: `Dim B` oder `Private B` gilt in allen Subs und Procedures des Moduls

**b) In einer Form**

- in den Allgemeinen Deklarationen der Form:
  `Dim C` gilt in allen Subs und Procedures der Form
- in einzelnen Prozeduren:
  `Dim D` gilt nur in der Prozedur, in der die Variable deklariert wurde **(lokale Variable)** Solche Variablen verlieren nach Beendigung der Prozedur ihre Gültigkeit.
- in einzelnen Prozeduren:
  `Static E` gilt nur in der Prozedur, in der die Variable deklariert wurde **(lokale Variable)** Solche Variablen behalten während der gesamten Laufzeit der Anwendung ihren Wert.

### Indizierte Variablen, Felder (Arrays)

Auch in Visual Basic gibt es Variablen mit Indizes (definiert einen "Vektor"):

| | |
|---|---|
| `Dim A(200)` | Index von 0 bis 200 |
| `Dim A(20 To 100)` | Index von 20 bis 100 |

### Mehrdimensionale Felder

```
Dim Matrix(1 to 5, 1 to 20)
Dim Matrix(100, 50)
```

### Dynamische Felder

| | |
|---|---|
| `Dim A()` | bedeutet, daß die Feldindizes zur Laufzeit nach Bedarf festgelegt und verändert werden können. Das geschieht durch die Anweisung |
| `Redim A(200)` | Dabei wird das Feld neu initialisiert, die vorangegangenen Werte gehen dabei verloren. |
| `Redim Preserve A(200)` | Die vorangegangenen Werte bleiben erhalten. |
| `Erase A` | Der Speicherbereich für A wird freigegeben. |

## 2.4 Message-Boxen

Jedes Programm besteht im allgemeinen aus den Teilen ***Eingabe – Verarbeitung – Ausgabe***. In Visual Basic gibt es eigene "Fenster", die sich besonders für Ausgaben bzw. Eingaben eignen. Zunächst wollen wir die "Ausgabefenster" besprechen:

Message-Boxen sind Ausgabefenster, deren Inhalt in einer Variablen Msg$ abgespeichert sein muß. Messageboxen dürfen max.1024 Zeichen enthalten, davon max. 256 in ununterbrochener Reihenfolge.

Messageboxen werden folgendermaßen aufgerufen:

```
MsgBox Msg$, Option%, Titel$
```

`Msg$` ist der Inhalt der Messagebox, also der Text, der in der Box erscheinen soll.

`Option%` ist eine Zahl, die durch Addition verschiedener Werte entsteht:

**1. Gruppe**

| | |
|---|---|
| 0 | Nur "OK" anzeigen. |
| 1 | "OK" und "Abbrechen" anzeigen. |
| 2 | "Abbrechen", "Wiederholen" und "Ignorieren" anzeigen. |
| 3 | "Ja", "Nein" und "Abbrechen" anzeigen. |
| 4 | "Ja", und "Nein" anzeigen. |
| 5 | "Wiederholen" und "Abbrechen" anzeigen. |

**2. Gruppe**

| | |
|---|---|
| 16 | Symbol anzeigen |
| 32 | Symbol anzeigen |
| 48 | Symbol anzeigen |
| 64 | Symbol anzeigen |

**3. Gruppe**

| | |
|---|---|
| 0 | Erste Schaltfläche ist Voreinstellung. |
| 256 | Zweite Schaltfläche ist Voreinstellung. |
| 512 | Dritte Schaltfläche ist Voreinstellung. |

**4. Gruppe**

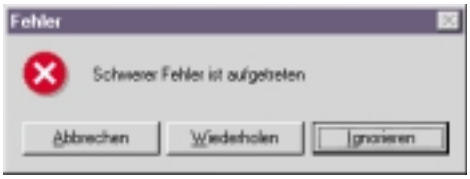
| | |
|---|---|
| 0 | An die Anwendung gebunden. Der Benutzer muß das Meldungsfeld beantworten, bevor er seine Arbeit an der aktuellen Anwendung wieder aufnehmen kann. |
| 4096 | An das System gebunden. Alle Anwendungen werden angehalten, bis der Benutzer das Meldungsfeld beantwortet. |

Beim Addieren von Zahlen zur Bildung eines Endwerts für das Argument Typ können Sie nur eine Zahl aus jeder Gruppe verwenden. Falls nicht angegeben, ist der Vorgabewert für Typ 0.

### Beispiel

```
MsgBox _
  "Schwerer Fehler ist aufgetreten", _
2+16+512+0, "Fehler"
```

liefert folgende Messagebox:




weil:
2 = Abbrechen,Wiederholen,Ignorieren
167 = Stop-Icon
512 = 3. Schaltfeld (Ignorieren) voreingestellt
0 = Box muß beantwortet werden, damit Programm fortsetzt

Für die Festlegung des Inhaltes muß man daher nur die Variable `Msg$` belegen, etwa wie folgt:


```
Sub menHilfe_Click ()
  Msg$ = "Hilfe wollen Sie?"
  MsgBox Msg$
End Sub
```

Das Ergebnis sieht folgendermaßen aus:

## 2.5 Input-Boxen

Dienen zur standardisierten Eingabe. Syntax:

```
Variable =
  InputBox(Msg$, Titel$, DefaultValue)
```

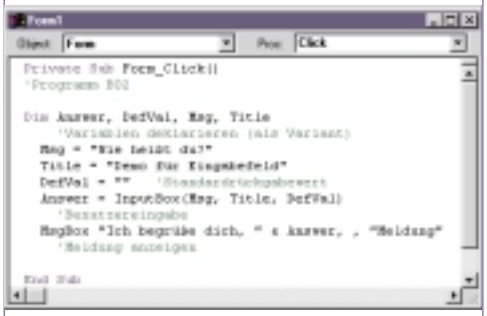
Dabei bedeuten: `Msg$`...Text der Eingabeaufforderung (also z.B. `"Geben Sie eine Zahl ein:"`) `Titel$`...Titelzeile des Eingabefensters; `DefaultValue`..Vorgabewert

**Beispiel**

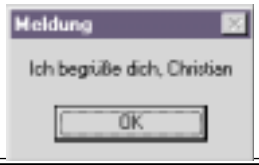
```
Answer = InputBox("Wie heißt Du?",
  "Demo für Eingabefeld", "")
```

liefert:


Wir wollen nun folgendes Programm als Anwendung der Ein- und Ausgabefenster ausprobieren:



Nach Drücken der ***OK-Taste*** erscheint:




**Beispiel 3:**
**Berechnung der Mehrwertsteuer (B03)**

### Entwicklung der Form

Der erste Schritt besteht immer in der Entwicklung der Form.

Wir ändern im Fenster "***Eigenschaften***" zunächst folgende Punkte:

`Caption` ("Überschrift"):

`Mehrwertsteuerberechnung`

`Name`:

FrmMWst

Als "`Name`" immer einen internen Namen angeben, auf den dann beim Programmieren Bezug genommen werden kann. Wichtig: Diese Namen sollte man sich merken oder notieren!

Es empfiehlt sich, standardisierte Namen zu vergeben. Für Formen ist etwa der Name "`frm...`" günstig.

Zunächst werden wir eine Pulldownmenü-Zeile entwickeln: Dazu wählen wir ***Extras – Menü - Editor*** (in der englischen Version ***Tools – Menu Editor***) oder einfacher: Mit der **rechten Maustaste** auf die Form klicken – im Kontextmenü findet man den Menüeditor!


Dann gibt man unter "`Caption`" den Menütitel an, der im fertigen Programm erscheinen soll. Ein `&` vor einem beliebigen Zeichen gestattet, diesen Menüpunkt mit der Tastenkombination ***«ALT» + Buchstabe*** aufzurufen. Der zweite "Menüpunkt" besteht aus einem einzelnen "-"-Zeichen; er bewirkt die Ausgabe eines Trennstriches im Pulldown-Menü:


Unter "`Name`" ist ein (interner) Name anzugeben; er dient später zur Programmierung.

*Tip*: Man sollte auch hier einheitliche Namen für die einzelnen Menüobjekte vergeben (etwa immer mit "`men`" beginnend), um später nicht in Verwirrung zu geraten!

| Menüpunkt (Caption) | interner Name |
|---|---|
| `&Hilfe` | `menHilfeTop` |
| `&Hilfe anzeigen` | `menHilfe` |
| `-` | `menStrich1` |
| `E&xit` | `menEnde` |

Ist man fertig mit einem Menüpunkt, so kann mit dem Schaltknopf "***Nächstes***" der nächste Menüpunkt angelegt werden.

Die "***Einrückungen***" bedeutet, daß "`Hilfe anzeigen`" und "`Exit`" Untermenüpunkte zum Menüpunkt "`Hilfe`" sind. Einrückungen sind mit den Cursor-Schaltknöpfen links neben dem Schaltknopf erreichbar.

*Wichtig*: Auch der "Linien-Menüpunkt" muß einen internen Namen erhalten!

**Deaktivieren bestimmter Menüpunkte:** Alle Menüeinträge können während der Laufzeit auch vorübergehend deaktiviert oder sogar versteckt werden. Beispiel:

```
menHilfe = Hidden
menHilfe = Visible
menHilfe = Disabled
menHilfe = Enabled
```

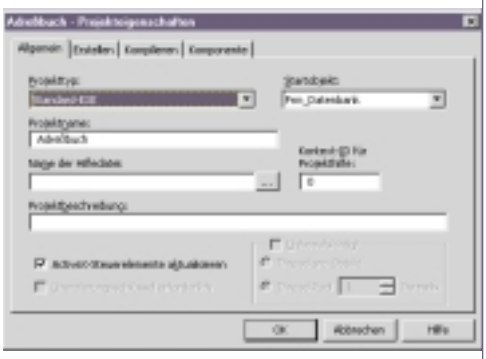
Für das "Angehakt-Sein" gibt es die Eigenschaft

```
menHilfe = Checked
```

### Die Projekteigenschaften im Menü [Projekt]-[Eigenschaften von *Projektname*]:

Viele interessante Dinge lassen sich in den Projekteigenschaften einstellen.

In der Karteikarte "***Allgemein***" muß ein Startobjekt angegeben werden – entweder eine Form oder eine Startprozedur `Sub Main()`. Weiters kann hier ein Projektname angegeben werden, der als Icon-Beschriftung dienen kann. Eine weitere Option besteht in der Angabe einer Help-Datei (Online-Hilfe) zum Programm.



## 2.6 Objekte – Objektorientierte Programmierung

Jede Form enthält eine Reihe von Steuerelementen: Bildlaufleisten, Schaltknöpfe (z.B. ***"OK"*** oder ***"Abbrechen"***) usw. Diese Elemente können alle mithilfe der Werkzeug-Leiste in die Form eingefügt werden.

An dieser Stelle soll erwähnt werden, daß alle Steuerelemente und auch die Form selbst sogenannte **OBJEKTE** darstellen. Visual Basic ist ein spezieller Typ der sogenannten "objektorientierten Programmiersprachen".

Jedes Objekt ist durch zwei Dinge charakterisiert:

- **Eigenschaften = Properties** (Variablen)
- **Verhalten** (Prozeduren, Funktionen)

So hat etwa eine Form die Eigenschaft "`Caption`". (In dieser Variablen ist die "Überschrift" abgespeichert, die man als Titelzeile einer Form sieht.) Man kann die-

se Eigenschaft entweder über das "Eigenschaften"-Fenster ändern oder aber vom Code aus:

```
Form1.Caption = "Mehrwertsteuerberechnung"
```

Die Variablen eines Objekts werden oft als **Attribute** bezeichnet. Jede Form hat aber auch die Möglichkeit, angezeigt zu werden. Dazu gibt es die Prozedur "`Show`", die ebenfalls auf diese Art und Weise aufgerufen werden kann:

```
Form1.Show
```

Die Prozeduren und Funktionen eines Objekts werden meist als **Methoden** bezeichnet.

Es gibt meist mehrere Objekte vom gleichen "Typ" (also zB mehrere Formen). Beide Objekte sind eigentlich nur spezielle "Variablen", deren "Objekttyp" generell festgelegt ist. Man sagt: Jedes Objekt ist eine **INSTANZ** seiner **KLASSE.**

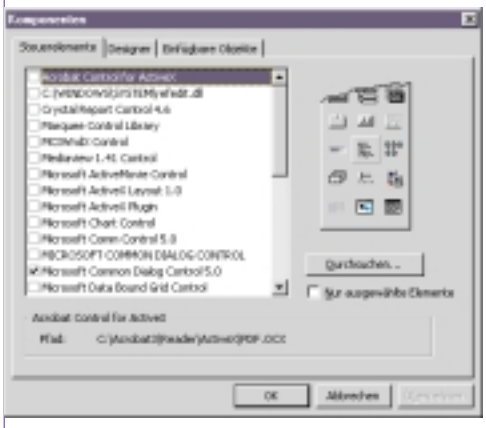
## 2.7 Die Werkzeugsammlung

***Wichtiger* Hinweis:** Nicht in allen Versionen sind alle beschriebenen Steuerelement-Werkzeuge sichtbar!

- Wenn der Zeiger ausgewählt ist, können Sie lediglich die Größe eines Steuerelements ändern oder es verschieben.
- Wenn irgendeines der anderen Werkzeuge ausgewählt ist, können Sie damit nur neue Steuerelemente zeichnen.
- Wenn Sie ein Steuerelement zeichnen, wird automatisch der Zeiger wieder ausgewählt.

### Standard-Version-Leiste

Die Werkzeugleiste ist für jedes Projekt individuell konfigurierbar: Mit [Projekt]-[Komponenten] können Sie ActiveX-Steuerelemente zu Ihrem Projekt hinzufügen!



Zeiger (Pointer): Wird benützt, um Steuerelemente ("controls") zu zeichnen, zu verschieben oder die Größe zu ändern. Nach der Plazierung eines Steuerelements auf einer Form ist automatisch wieder der Zeiger aktiv. Der Zeiger selbst ist kein Steuerelement.

Die Steuerelemente können in drei Gruppen gegliedert werden:

- Integrierte Steuerelemente, die in der .exe-Datei von Visual Basic enthalten sind (zum Beispiel das Rahmen-Steuerelement oder das Befehlsschaltfläche-Steuerelement)
- ActiveX-Steuerelemente, die als eigenständige Dateien mit der Erweiterung `.ocx` zu finden sind (zum Beispiel das Tabelle-Steuerelement, das Kombinationsfeld-Steuerelement usw.)
- Einfügbare Objekte (etwa OLE-Container und dergleichen)

### a) Integrierte Steuerelemente

PictureBox. Kann Grafiken (Bitmap `*.BMP`, Icon `*.ICO`, Metafile `*.WMF`, `*.GIF`, `*.JPG`) darstellen. Kann auch andere Steuerelemente enthalten und Text anzeigen.

Label (Bezeichnungsfeld). Für Texte, die nicht vom User geändert werden sollen (Überschriften etc.).

TextBox (Eingabefeld, "Textfeld"). Hierin befindet sich Text, der vom User geändert werden kann/soll.

Frame (Rahmen). Dient zur Gruppierung von Steuerelementen.

CommandButton (Befehlsschaltfläche). Erzeugt eine Schaltfläche, deren Anklicken ein Unterprogramm auslöst.

CheckBox (Auswahlfeld). Hier kann eine wahr/falsch-Möglichkeit ausgewählt werden bzw. eine Auswahl aus mehreren Punkten vorgenommen werden. (Das Anklicken keines oder mehrerer Felder ist möglich.)

OptionButton (Optionsfeld). Hier kann der User nur eine Möglichkeit auswählen.

ComboBox (Kombinationslistenfeld) = Kombination aus einem Listenfeld und einem Textfeld Der User kann entweder eine Eingabe in den Textfeldteil schreiben oder aus dem Listenteil eine Möglichkeit wählen.

ListBox (Listenfeld). Scrollbare Liste mit Elementen, aus denen der User eines auswählen kann.

HScrollBar (horizontal scroll bar, horizon

tale Bildlaufleiste). Wird verwendet, um rasch eine lange Liste durchlaufen zu lassen, die aktuelle Position auf einer Skala anzugeben oder als Eingabemöglichkeit für Geschwindigkeiten oder Anzahlen.

VScrollBar (vertical scroll bar, vertikale Bildlaufleiste). Siehe HScrollBar.

Timer (Zeitgeber). Für zeitliche Steuerungen, zur Laufzeit unsichtbar.

DriveListBox (Laufwerkslistenfeld). Zeigt gültige Laufwerksbezeichnungen an.

DirListBox (directory list box, Verzeichnislistenfeld). Zeigt Verzeichnisse und Pfade an.

FileListBox (Dateilistenfeld): Zeigt Dateiliste an.

Shape (Figur). Damit können Figuren wie Rechteck, gerundetes Rechteck, Quadrat, abgerundetes Quadrat, Ellipse oder Kreis dargestellt werden.

Line (Linie). Zur Entwicklungszeit können Linien verschiedener Stärke, Farbe und Art gezeichnet werden.

Image. Stellt eine Grafik (Bitmap `*.BMP`, Icon `*.ICO`, Metafile `*.WMF`, `*.GIF`, `*.JPG`) in einer Form dar und verhält sich wie eine Befehlsschaltfläche, wenn darauf geklickt wird.

Data. Für Zugriff auf Daten in Datenbanken.

OLE Container. Damit können Objekte anderer Windows-Applikationen (Excel, Word usw.) in Ihre Visual-Basic-An-

wendung eingebettet oder verbunden werden.

**b) Standard-ActiveX-Steuerelemente**

Die folgenden Steuerelemente sind bereits in der Standard-Edition von Visual Basic 5.0 enthalten! In der Professional und Enterprise-Version stehen wesentlich mehr Steuelemente dieser Art zur Verfügung.


CommonDialog (Standarddialog). Wird benützt, um Dialogboxen zu erzeugen, wie sie zum Beispiel zum Datei-Öffnen, Datei-Speichern oder Auswahl von Schriftarten und Farben benützt werden. CommonDialog ist ein "custom control".


DBList (data-bound list box). Listenfeld mit erweiterten Datenbankmöglichkeiten.


DBCombo (data-bound combo box). Kombinationslistenfeld mit erweiterten Datenbankmöglichkeiten.


DBGrid (data-bound grid). Wird verwendet, um eine Tabelle darzustellen, in welcher Daten manipuliert werden können. Unterscheidet sich vom Standard-Gitternetz-Objekt durch die erweiterten Datenbankzugriffseigenschaften.

**b) ActiveX-Steuerelemente in der Professional und Enterprise-Edition**


SSTab (Registerkartendialog). Mit diesem Objekt können "Karteikarten" erstellt werden, mit denen eine komplexe Menüauswahl möglich ist. Vergleiche das Menü Extras-Optionen in MS-Word:


TabStrip (Register; ähnliche Funktionsweise wie SSTab-Dialogfeld: Karteikartenähnliche Darstellung)


Rich Text Box: Stellt ASCII- und RTF-Texte (RTF = rich text format) dar, kann auch Dateien öffnen. Optimal als "Mini-Editor" verwendbar.


ToolBar (Werkzeugleiste: beinhaltet einige Buttons, die bestimme Aktionen auslösen)


StatusBar (Statuszeile: fügt eine Statuszeile – meist am unteren Fensterrand – ein)


ProgressBar (zeigt den Fortschritt einer Aktion an, zB Speichern)


TreeView (stellt eine Baumstruktur dar)


ImageList (besteht aus einer Liste von Grafiken = "Images")


ListView (damit können Listen, etwa Dateilisten, übersichtlicher dargestellt werden – so wie im Windows 95-Explorer)


Slider (Regler): kann mit der Maus hin- und herverschoben werden


Winsock


WebBrowser

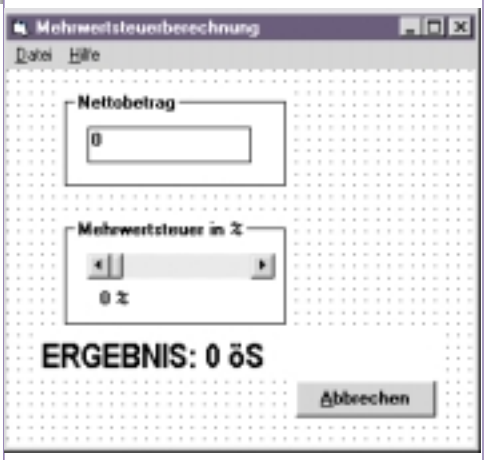

MAPISession


MAPIMessages


MMControl (Multimedia-Steuerelement)


Sysinfo: Überwacht eine Reihe von Parametern des Betriebssystems und benachrichtigt Ihre Anwendungen bei Änderungen.

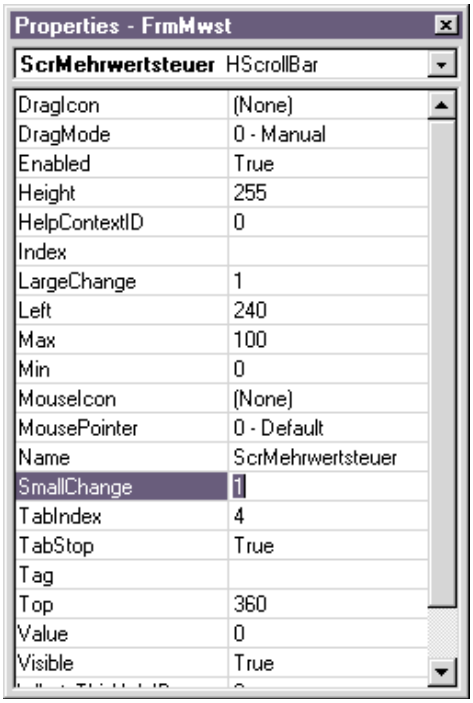
Damit plaziert man die notwendigen Steuerelemente in das Form-Fenster, sodaß es etwa wie folgt aussieht:




- zwei Rahmen
- im oberen Rahmen: ein Textfeld
- im unteren Rahmen: eine horizontale Bildlaufleiste und ein Bezeichnungsfeld
- ein zweites Bezeichnungsfeld extra
- eine Befehlsschaltfläche

*Anmerkung*: Jedes Element kann durch "Ziehen" in seiner Größe verändert werden!

Nun geht es darum, die Eigenschaften jedes Steuerelements in der "Eigenschaften"-Tabelle einzustellen:




- oberer Rahmen

  `Caption` *Nettobetrag*

  `Name` *RahNettobetrag*

- unterer Rahmen

  `Caption` *Mehrwertsteuer in %*

  `Name` *RahMehrwertsteuer*

- Horizontale Bildlaufleiste

  `Max` *100*
  **(Achtung: Maximal möglicher Wert 32767!)**

  `Min` *0*

  `Name` *ScrMehrwertsteuer*

- Schaltknopf

  `Caption` *&Abbrechen*
  (Das &-Zeichen bedeutet, daß der Schaltknopf auch durch die Tastenkombination «AltíA»ausgelöst werden kann.)

  `Name` *CmdAbbrechen*

  `Default` *True*
  (das hat zur Folge, daß dieser Knopf beim Programmstart aktiv ist)

`Cancel` *True*
(dies bedeutet, daß dieser Knopf dieselbe Auswirkung wie das Drücken der ESC-Taste haben soll)

- Bezeichnungsfeld im unteren Rahmen

  `Caption` *0 %*

  `Name` *BezMehrwertsteuer*

- Bezeichnungsfeld außerhalb

  `Caption` *Ergebnis: 0 %*

  `Name` *BezErgebnis*

  `Font` *Arial Narrow, 18 pt, Fett*

- Textfeld zur Eingabe des Betrags:

  `Text` *0*

  `Name` *EinNettobetrag*

An dieser Stelle sollte man sich eine **Tabelle** anlegen, in der die internen Namen aller Objekte in der Form enthalten sind! Es ist nicht möglich, sich alle Objektnamen zu merken, wenn man größere Projekte mit einigen 100 Objekten erstellt!

## 2.8 Ereignisgesteuerte Programmierung

Damit wäre die Erstellung der Form abgeschlossen. Nun geht es an die eigentliche Programmierung.

Dazu einige Vorbemerkungen:

1 Alle Visual Basic-Programme bestehen aus abgetrennten Programmteilen = Unterprogrammen. Man unterscheidet **Prozeduren** (`Sub`) und **Funktionen** (`Function`).

2 **Ereignisgesteuerte Programmierung:** Jede Prozedur beginnt mit dem Wort `Sub` und endet mit `End Sub`. Nach dem Wort `Sub` steht der Name der Prozedur. Dieser Name ist bei **Ereignisprozeduren** nicht frei wählbar, sondern besteht aus dem Namen des Objekts und dem Namen der Aktion (des **Ereignisses = Events**), bei der das Unterprogramm ausgelöst werden soll. Bei Prozeduren, die nicht von einem Ereignis abhängen, sondern anders (etwa durch Aufruf in einer anderen Prozedur) gestartet werden, ist der Name frei wählbar.

Zuerst programmieren wir den Schalter "Abbrechen": Die Prozedur muß daher heißen "`Private Sub CmdAbbrechen_Click()`". Hier soll das Programm beendet werden. Das geschieht mit dem Befehl `Unload`.

(Übrigens: Kommentare beginnen in VB mit einem einfachen Apostroph-Zeichen ').

```
Private Sub CmdAbbrechen_Click ()
' Was passiert bei Anklicken der
' "Abbrechen"-Taste
  Unload MWSt
End Sub
```

Mit der Anweisung `Unload` wird die Startform entladen, das heißt, sie wird aus dem Arbeitsspeicher entfernt. Statt des Namens der Form kann in diesem Fall auch

```
Unload Me
```

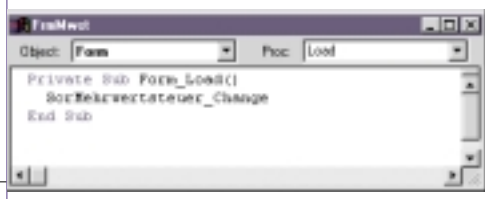
stehen, wobei `Me` für die aktuelle Form steht.

Mit dem Pulldown-Menü Ausführen – Starten kann das Programm gestartet werden.

*Probieren Sie aus, ob die* ABBRECHEN*-Taste funktioniert*!

Eine der wichtigsten Ereignisprozeduren ist `FORM_LOAD`. Dieses wird immer dann aufgerufen, wenn die Form geladen wird – also beim ***Programmstart*!**

In diesem Fall soll dieses Programm eine andere Ereignisprozedur auslösen, die den Sinn hat, die Bildlaufleiste immer zu aktualisieren. Dies sieht dann so aus:




Dabei haben wir noch nicht festgestellt, was das "Aktualisieren" der Bildlaufleisten eigentlich bewirken soll. Die Prozedur `ScrMehrwertsteuer_Change` wird also noch zu programmieren sein.

Unter "Aktualisieren" versteht man, daß das Programm auf jede Veränderung der Bildlaufleiste entsprechend reagieren soll. In unserem Fall soll sich die Bezeichnung unter der Bildlaufleiste ändern.

Wir weisen daher der Eigenschaft `Caption` des Bezeichnungsfeldes unter der Bildlaufleiste den Wert der Bildlaufleiste zu:

```
BezMehrwertsteuer.Caption =
ScrMehrwertsteuer.Value & " %"
```

"`Value`" war offenbar mit der Bildlaufleiste fest verbunden. Eine derartige "Eigenschaft" wird vom Objektnamen durch einen Punkt getrennt. Durch das Verkettungszeichen "&" wird an den Wert noch das "%"-Zeichen angehängt, weil wir ja möchten, daß als Beschriftung z.B. "37 %" erscheint.

Schließlich muß eine erneute Berechnung erfolgen, da sich der Prozentwert ja geändert hat. Für die Berechnung benötigen wir allerdings eine neue Prozedur, die wir `Private Sub Neuberechnung()` nennen wollen.

Völlig neue Prozeduren – wie unsere Berechnungsprozedur Neuberechnung – können nur mit ***Einfügen – Prozedur*** (***Insert – Procedure***) erstellt werden. Diese Vorgangsweise werden wir bei der eigentlichen Berechnung des Endbetrages anwenden.

Bei dieser Prozedur müssen wir das Problem lösen, wie wir aus den Eingabeobjekten (der Bildlaufleiste für die Mehrwertsteuer und dem Textfeld für den Betrag) die eigentlichen Zahlenwerte "herauskitzeln".

Für den Wert einer Bildlaufleiste kennen wir schon die Eigenschaft Value, die wir einer Variable `mwstsatz` zuweisen können:

```
mwstsatz = ScrMehrwertsteuer.Value
```

Für die Eingabe des Betrags wurde ein Textfeld verwendet. (Es ist nicht sehr sinnvoll, den Grundbetrag durch eine Bildlaufleiste einzuschränken. Außerdem können durch Bildlaufleisten nur schwer "Groschenbeträge" eingegeben werden.)

Die Eigenschaft Text beinhaltet als Wert zwar den eingegebenen Text, allerdings als String. Die Funktion `Val()` wandelt eine Zeichenkette (String) in eine Zahl um:

```
nettobetrag = Val(EinNettobetrag.Text)
```

Somit können wir – die mathematische Formel für Prozentrechnung vorausgesetzt – die Prozedur Neuberechnung entwerfen:

```
Private Sub Neuberechnung()
  Dim nettobetrag As Single
  Dim mwstsatz As Single
  Dim steuer As Single
  Dim erg
  nettobetrag =  Val(EinNettobetrag.Text)
  mwstsatz = ScrMehrwertsteuer.Value
  steuer = nettobetrag * mwstsatz / 100
  erg = "ERGEBNIS: " & Str$(steuer) & " öS"
  BezErgebnis.Caption = erg
End Sub
```

Die Funktion `Str$()` wandelt eine Zahl in eine Zeichenkette (string) um.

**Formatierte Ausgabe:** Oft ist eine Ausgabe wie "2.3859234E-2" nicht sehr übersichtlich. Um eine genormte Ausgabe (etwa: 2 Kommastellen, Tausender-Trennzeichen) zu erreichen, kann man die Funktion `Format$` verwenden.

Beispiel:

```
MsgBox "Ergebnis:" & Format$(erg,
"#.##0.00")
```

Der zweite Parameter in der Funktion `Format$` ist der "Format-String", Dabei steht ein "#" für ein Zeichen, das bei Bedarf (bzw. Vorhandensein) angezeigt wird, eine "0" für eine Stelle, die immer angezeigt werden muß.

Beispiele für das verwendete Format "#,##0.00":

| | | |
|---|---|---|
| 12,3452542 | angezeigt als | 12,34 |
| 19342,1 | angezeigt als | 19.342,10 |
| 0,1234 | angezeigt als | 0,12 |

Wir haben auch noch die beiden Pulldown-Menüpunkte "***Ende***" und "***Hilfe***". Den Menüpunkt Ende können wir ganz einfach behandeln:

```
Private Sub menEnde Click ()
  Unload MWSt
End Sub
```

Damit wird das Programm beendet.

Beim Menüpunkt "***Hilfe***" soll ein ganz einfacher Hilfetext erscheinen. Das realisiert man am einfachsten mit der bereits bekannten **Message-Box.**

Möchte man mehrere Ausgabezeilen in der Messagebox erreichen, so kombiniert man Texte mit Zeilenschaltungen.

In Windows wird für die Behandlung von Zeichen die sogenannte ANSI-Code-Tabelle verwendet (siehe Anhang B!). Diese besteht aus 256 Zahlen, von denen jede Zahl für ein darstellbares Zeichen steht.

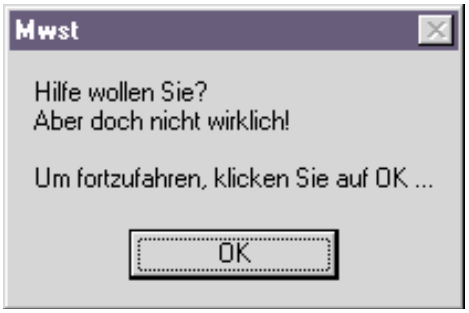
Die Werte 08, 09, 10 und 13 stehen nicht für Zeichen, sondern verändern die Ausgabe:

| | | | |
|---|---|---|---|
| 08 | BS | back space | Rückwärtsschritt und Zeichen löschen |
| 09 | HT | horizontal tab | waagrechter Tabulatorsprung |
| 10 | LF | line feed | Zeilenvorschub |
| 13 | CR | carriage return | Wagenrücklauf |

Es ist daher sinnvoll, eine Variable `NZ$` (für "neue Zeile") zu definieren, die aus Zeilenvorschub und Wagenrücklauf besteht. Für spezielle ANSI-Zeichen gibt es dafür die Funkion `Chr$()`. Das sieht dann etwa so aus:

```
Private Sub menHilfe Click ()
  NZ$ = Chr$(13) + Chr$(10)
  Msg$ = "Hilfe wollen Sie?" + NZ$
  Msg$ = Msg$ + "Aber doch nicht wirklich!"
  Msg$ = Msg$ + NZ$ + NZ$
  Msg$ = Msg$ + "Um fortzufahren, klicken"
  Msg$ = Msg$ + "Sie auf OK ..."
  MsgBox Msg$
End Sub
```

**Ergebnis**




Eine bessere Möglichkeit besteht darin, die vordefinierte Konstante `vbCrLf` zu verwendet, die als

```
vbCrLf = Chr$(13) + Chr$(10)
```

definiert ist.

Das obige Programm lautet dann:

```
Private Sub menHilfe_Click ()
  Msg$ = "Hilfe wollen Sie?" + vbCrLf +
"Aber doch nicht wirklich!"
  Msg$ = Msg$ + + vbCrLf + vbCrLf
  Msg$ = Msg$ + "Um fortzufahren, klicken
Sie auf OK ..."
  MsgBox Msg$
End Sub
```

### Festlegen der Aktivierreihenfolge mit dem Tabulator-Index

Durch Betätigen der ***«TAB»***-Taste springt der Cursor (im englischen Original übrigens "Caret" genannt) zum nächsten aktivierbaren Feld.

Angenommen, es gibt mehrere Textfelder zur Eingabe. Dann möchte man natürlich, daß der Cursor beim Programmstart auf dem 1. Textfeld steht; weiter geht es mit der ***«TAB»***-Taste. Dazu benützt man die Eigenschaft "TabIndex": in der Reihenfolge ihres Anwählens mit ***«TAB»*** sollen die Steuerelemente mit 0, 1, 2 usw. durchnumeriert werden.

**Rechtsbündige Textfelder:** Sind nur möglich, wenn man die Eigenschaft MultiLine auf True setzt. Diese Eigenschaft erlaubt mehrzeilige Eingaben.

Möchte man beim Programmstart bestimmte Elemente nicht sehen, so setzt man die Eigenschaft Visible auf False.

## 2.9 Lauffähiges EXE-Programm

Unter ***Datei – Exe-Datei erstellen*** ist es möglich, aus dem Visual Basic-Projekt eine selbständig unter Windows lauffähige `EXE`-Datei zu erstellen. Der Kunde benötigt nun nicht mehr die Visual Basic-Entwicklungsumgebung, sondern nur mehr die erstellte Datei sowie die Datei ***VBRUN400.DLL*** (Runtime-Datei). Diese Datei darf ohne Verletzung der Urheberrechte an Klienten weitergegeben werden.

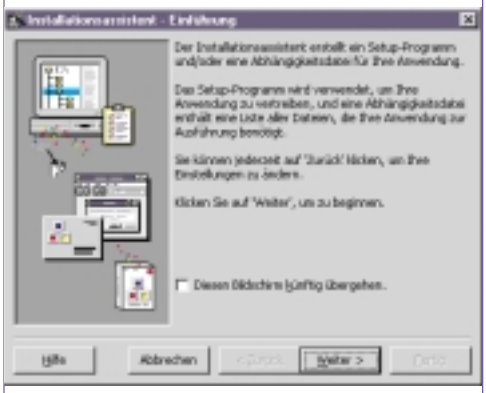
Die Namen der Laufzeitbibliotheken in den einzelnen Versionen:

| | |
|---|---|
| `VBRUN100.DLL` | (Version 1.X) |
| `VBRUN200.DLL` | (Version 2.0) |
| `VBRUN300.DLL` | (Version 3.0) |
| `VB40016.DLL` | (Version 4.0, 16 bit) |
| `VB40032.DLL` | (Version 4.0, 32 bit) |
| `MSVBVM50.DLL` | (Version 5.0, 32 bit) |

## 2.10 Erstellen eines Setup-Programms

Wenn Visual Basic-Programme weitergegeben werden sollen, so empfiehlt sich die Erstellung einer Setup-Routine, die automatisch alle nötigen Dateien in die richtigen Verzeichnisse installiert. Dazu gibt es den Installations-Assistenten, der sich in der Gruppe Visual Basic befindet:

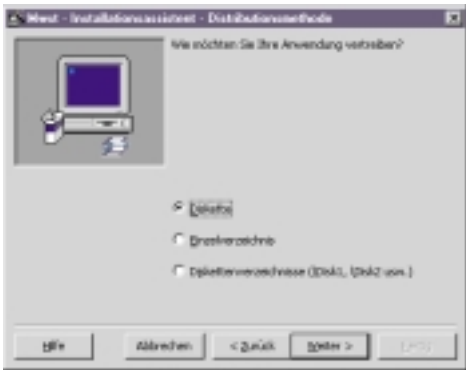

Doppeltes Anklicken startet ein spezielles Visual-Basic-Programm, welches alle erforderlichen Schritte zur Packen der Dateien auf eine Diskette durchführen kann:



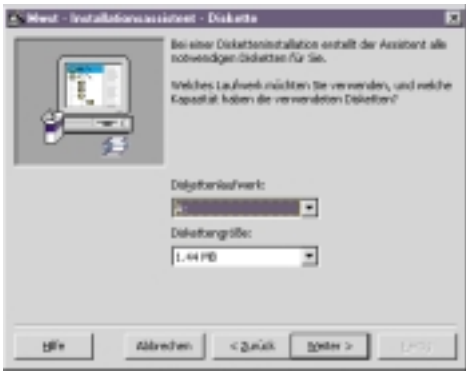

Zunächst müssen Sie den Pfad der Projektdatei angeben. In den Optionen können Sie auswählen, ob Sie ein komplettes Installationsprogramm (lokal oder fürs Internet) oder nur eine Abhängigkeitsdatei erstellen lassen wollen. Eine solche Abhängigkeitsdatei enthält eine Liste aller Dateien, die zur Ausführung des Projektes notwendig sind.


Nun legen Sie fest, auf welchem Weg Sie Ihre Anwendung vertreiben wollen: Als Disketten, als CD-ROM (hier eignet sich

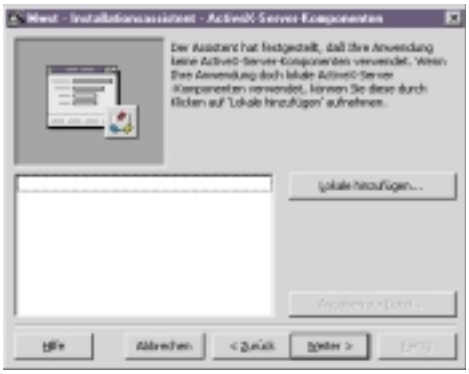
ein Einzelverzeichnis) oder als Diskettenverzeichnisse.



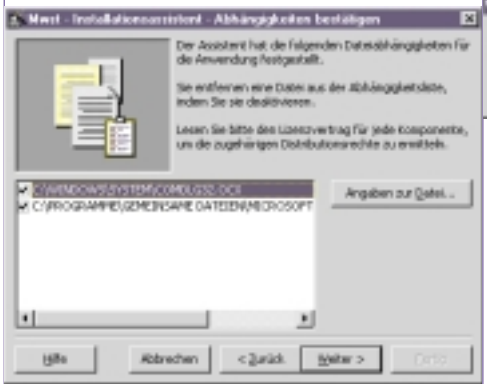
Nun wählen Sie den Diskettentyp aus:



Falls nötig, können noch ActiveX-Server-Komponenten hinzugefügt werden:

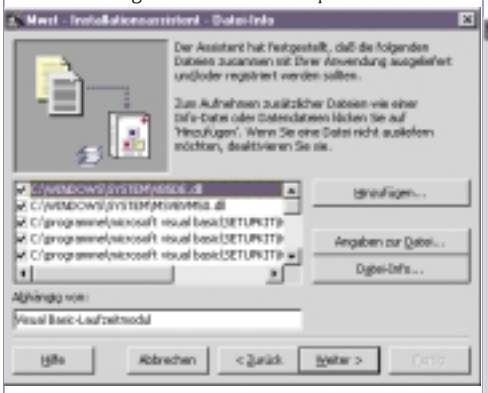


Im folgenden Dialogfenster müssen Sie noch bestätigen, daß bestimmte zusätzliche Dateien (etwa OCX-Dateien für verwendete ActiveX-Steuerelemente) dazugefügt werden.

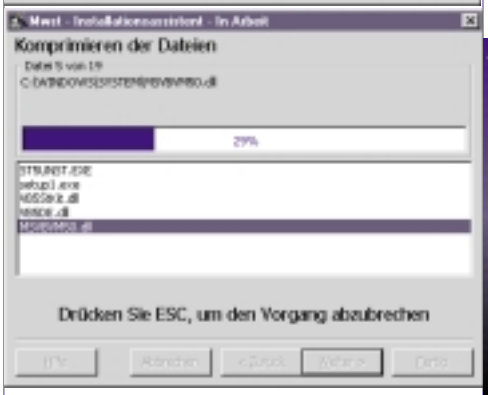


Der Installations-Assistent verarbeitet dann alle Informationen und zeigt an, welche Dateien zusammen mit Ihrer Anwendung mitgeliefert und registriert werden sollten:

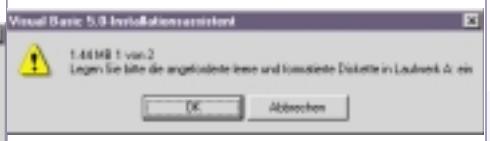
Klicken Sie auf ***"Fertig"***; dann werden alle notwendigen Dateien komprimiert und



auf Diskette kopiert. Beim Komprimieren erscheinen Zeilen wie die folgende:

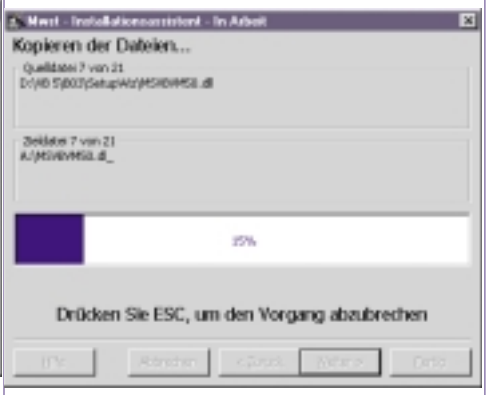


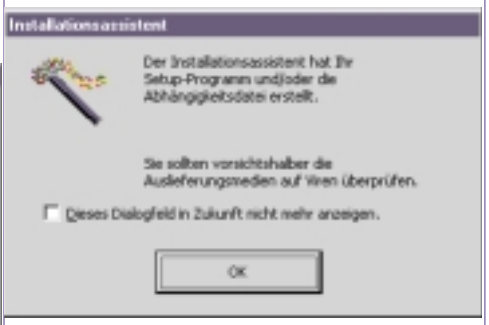
Das Programm verwendet die `COMPRESS.EXE`-Routine von Microsoft, um die Dateien zu komprimieren. Dann folgt die Meldung:



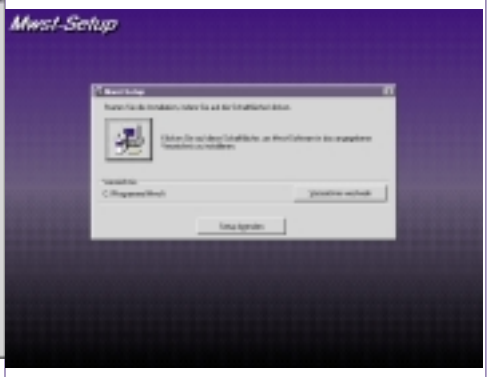
Legen Sie die angeforderten Disketten ins Laufwerk ein. Der Installations-Assistent erstellt dann die nötigen Startdisketten:

Nach Abschluß dieses Vorgangs erscheint folgende Meldung:





Nun befindet sich auf Diskette ein funktionsfähiges Windows-Programm mit dem Dateinamen `SETUP.EXE`, welches folgenden Bildschirm liefert:




## 2.11 Verzweigungen

### 1. Einfache Verzweigungen mit IF

Soll es an einer Stelle zwei Möglichkeiten des weiteren Programmablaufs geben, so gibt es in BASIC die Möglichkeit einer "Verzweigung". Syntax:

```
IF (Bedingung 1) THEN
  (Befehl)
  (Befehl)
   ...
ELSEIF (Bedingung 2) THEN
  (Befehl)
  (Befehl)
   ...
ELSE
  (Befehl)
  (Befehl)
  ...
END IF
```

Beispiel:

```
IF a 10 THEN
  Print "Die Zahl hat eine Stelle."
ELSEIF a = 10 And a 100 THEN
  Print "Die Zahl hat zwei Stellen."
ELSE
```

```
  Print "Die Zahl hat mehr als zwei
Stellen."
END IF
```

### 2. Mehrfache Verzweigungen mit SELECT CASE

Beispiel: Dieses Beispiel findet heraus, ob ein Großbuchstabe, ein Kleinbuchstabe, eine Zahl oder ein anderes Zeichen eingegeben wurde.

```
Private Sub Form Click ()
  Dim Msg$, UserInput
     ' Variablen deklarieren.
  Msg$ = "Geben Sie einen Buchstaben oder"
  Msg$ = Msg$ + "eine Zahl "
  Msg$ = Msg$ + " zwischen 0 und 9 ein."
  UserInput = InputBox(Msg$)
       ' Benutzereingabe anfordern.

  If Not IsNumeric(UserInput) Then
       ' Buchstabe oder Zahl?
    If Len(UserInput)  0 Then
       Select Case Asc(UserInput)
          ' Falls Buchstabe
          ' Muß Großbuchstabe sein.
       Case 65 To 90
     Msg$ = "Sie haben den Großbuchstaben '"
     Msg$ = Msg$ + Chr(Asc(UserInput))
     Msg$ = Msg$ + "' eingegeben."
            ' Muß Kleinbuchstabe sein.
       Case 97 To 122
     Msg$ = "Sie haben den Kleinbuchstaben '"
     Msg$ = Msg$ + Chr(Asc(UserInput))
     Msg$ = Msg$ + "' eingegeben."
       Case Else ' Muß etwas anderes sein.
     Msg$ = "Ihre Eingabe ist weder ein"
     Msg$ = Msg$ + " Buchstabe noch eine"
     Msg$ = Msg$ + "  Zahl."
       End Select
    End If     'Len(UserInput)...

  Else   'If Not IsNumeric ...
    Select Case CDbl(UserInput)
     ' Falls eine Zahl.
    Case 1, 3, 5, 7, 9  ' Ist ungerade.
       Msg$ = UserInput + " ist ungerade."
    Case 0, 2, 4, 6, 8  ' Ist gerade.
       Msg$ = UserInput + " ist gerade."
    Case Else ' Außerhalb des Bereichs.
       Msg$ = "Die eingegebene Zahl liegt
       Msg$ = Msg$ + "außerhalb des "
       Msg$ = Msg$ + "geforderten Bereichs."
    End Select
  End If      'If Not IsNumeric ...

  MsgBox Msg$ ' Meldung anzeigen.
End Sub
```

## 2.12 Wiederholungsstrukturen ("Schleifen")

Soll ein Programmteil mehrmals durchgeführt werden, so verwendet man sogenannte "Schleifen", die eine wiederholte Durchführung automatisch gestatten:

### 1. FOR – NEXT und FOR EACH – NEXT: "Zählschleife"

Die Anzahl der Wiederholungen muß bekannt sein.

**Beispiel 1**

```
For I = 1 to 5
  Print " Test " + Str$(I)
Next I
```

```
Ergebnis:
Test 1
Test 2
Test 3
Test 4
Test 5
```

**Beispiel 2**

```
For I = 5 to 1 Step -1
  Print " Test " + Str$(I)
Next I
```

```
Ergebnis:
Test 5
Test 4
Test 3
Test 2
Test 1
```

Ein "Verwandter" der `For-Next`-Schleife ist die `For Each-Next`-Schleife. Diese eignet sich besonders für die Abarbeitung einer Objektsammlung:

```
Gefunden = False' Variable
For Each MyObject In MyCollection
  ' Jedes Objekt in MyCollection wird
  ' überprüft.
  If MyObject.Text = "Hallo" Then
     ' Falls Text gleich "Hallo".
     Gefunden = True
     ' Setze Variable Gefunden auf True.
     Exit For
     ' Ausstieg aus der Schleife.
  End If
Next
```

### 2. DO – LOOP UNTIL: "Schwanzgesteuerte Schleife" (Abbruchbedingung am Ende)

```
Private Sub Form Click ()
  Dim Antwort As String
  Dim Msg As String
    ' Variablen deklarieren.
  Msg = "Bitte eine Zahl zwischen"
  Msg = Msg + "1 und 9 eingeben!"
  Do
    Antwort = InputBox(Msg)
  Loop Until Antwort  1 And Antwort 9
End Sub
```

`Do – Loop` ist die strukturierteste Art, Schleifen zu programmieren. Schleifen eignen sich hervorragend zum Abfangen von Eingabefehlern, wie folgendes Beispiel zeigt:

```
Sub Uprog ()
  Dim A As Integer
  Dim Eingabe As String
  Dim Msg As String, ErrMsg As String
  Msg = "Geben Sie bitte eine Zahl ein!"
  ErrMsg = "Eingabefehler! "
  ErrMsg = ErrMsg & "Bitte wiederholen"
  ErrMsg = ErrMsg & "Sie die Eingabe"

  Do
    Eingabe =  InputBox(Msg)
    If Not IsNumeric(Eingabe) Then
      MsgBox ErrMsg, 16, "Fehlermeldung"
    End If
  Loop Until IsNumeric(Eingabe)

End Sub
```

Die Funktion `IsNumeric(a)` liefert `true` (wahr), wenn `a` numerisch (also eine Zahl) ist, ansonsten `false` (falsch).

### 3. WHILE – WEND: "Kopfgesteuerte Schleife" (Abbruchbedingung am Anfang)

```
While (Bedingung)
  (Anweisungsblock)
Wend
```

**Zur Unterscheidung der Schleifenarten**

| | For – Next | Do – Loop Until | While – Wend |
|---|---|---|---|
| Abbruchbedingung | Durchlaufzahl | am Ende | am Anfang |
| Mindestdurchläufe | Anzahl vorgegeben | 1 | 0 |
| Maximale Durchlaufzahl | vorgegeben | "unbegrenzt" | "unbegrenzt" |

## 2.13 Ausdrucken

### 1. Ausdrucken über das Drucker-Objekt

Möchte man Text an den Drucker schikken, so geschieht das mit dem sogenannten **Printer Object.** Alle Dateien, die ausgedruckt werden sollen, werden an das Printer Object übergeben und danach ausgedruckt (vom Windows-Druck-Manager).

Zuerst müssen die Ausdruck-Eigenschaften festgelegt werden.

Beispiel:

```
Printer.FontName = "Times New Roman"
  'fett auf TRUE gesetzt
Printer.FontBold = -1
Printer.Print "Das ist Text auf dem Drucker"
Printer.NewPage
Printer.Print "Das ist Seite 2"
Printer.EndDoc
```

Zunächst werden die Daten in den Spooler geschrieben, mit der Anweisung `EndDoc` abgeschickt.

### 2. Ausdrucken einer ganzen Form

```
Form1.PrintForm
```

Wichtig: Auch versteckte Texte werden in diesem Fall mit ausgegeben. Sollte sich Grafik in der Form befinden, so muß die Form zunächst neu gezeichnet werden. Das geschieht mit dem Befehl:

```
Form1.AutoRedraw = -1
```

## 2.14 Arbeiten mit mehreren Forms

Zunächst müssen alle gewünschten Formen mit dem Befehl

```
Load Form1
```

geladen werden. Um eine Form sichtbar zumachen, verwendet man die Methode `Show`, um sie wieder zu "verstecken", die Methode `Hide`.

```
Form1.Show
Form2.Hide
```

Am Ende des Programms müssen alle Formen in gewohnter Form mit

```
Unload Form1
```

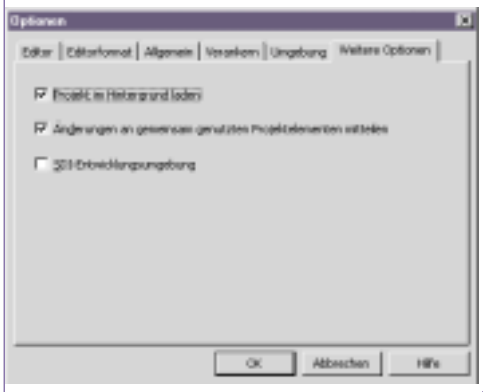
wieder aus dem Speicher entfernt werden – wichtig dabei: Die Startform **zuletzt!**

### SDI- und MDI-Forms

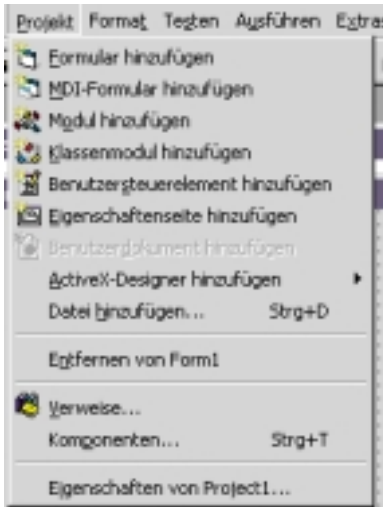
SDI = "Single Document Interface"

MDI = "Multiple Document Interface".

Ein MDI-Formular kann mehrere "Kind-Fenster" enthalten, ein SDI-Formular nicht. Man kann die VB-Entwicklungsumgebung so anpassen, daß entweder MDI-Formulare standardmäßig erzeugt werden oder SDI-Formulare. Das geschieht im Menüpunkt [Extras]-[Optionen]: Für die SDI-Entwicklungsumgebung markieren Sie bitte das entsprechende Kontrollkästchen.




Hier geht es um das Arbeiten mit "Fenstern innerhalb von Fenstern". Dabei muß eine **MDI-*Parent*-*Form*** definiert werden. Das geschieht mit [Projekt]-[MDI-Formular hinzufügen]:




Die MDI-Parent-Form ist die übergeordnete Form, die alle anderen enthält.

Die untergeordneten Formen werden als **MDI-*Child*-*Forms*** bezeichnet. Sie können innerhalb einer Parent-Form frei bewegt werden, aber nicht darüber hinaus. Wird das übergeordnete Fenster geschlossen, so verschwindet auch die Child Form.

Für Child Forms muß die Eigenschaft MDIChild = -1 (true) gesetzt werden.

## 2.15 Fehlerbehandlung in Visual Basic

Sie können auffangbare Fehler mit der **On Error-Anweisung** und der **Err-Funktion** testen und darauf reagieren. Tritt ein Fehler auf, so wird die Art des Fehlers in einer ***Fehlernummer*** gespeichert, die mit der Funktion `Err` abgefragt werden kann. (Eine Tabelle, was die einzelnen Nummern bedeuten, finden Sie auf den folgenden Seiten!)

Syntax:

```
On Error Goto Sprungmarke
```

Wenn Sie die On Error-Anweisung nicht verwenden, wird jeder auftretende Laufzeitfehler als fataler Fehler behandelt. Das heißt, Visual Basic erzeugt eine Fehlermeldung und unterbricht die Programmausführung.

Bei der Verwendung von **On Error Resume Next** wird die Programmausführung mit der Anweisung fortgesetzt, die sich unmittelbar an die Anweisung anschließt, die den Laufzeitfehler verursacht hat. Auf diese Weise kann Ihr Programm trotz eines Laufzeitfehlers fortgeführt und die Ursache für den Fehler nachträglich ermittelt werden. Ferner können Sie dadurch die Fehlerbehandlung direkt an der Fehlerstelle in der Prozedur durchführen, anstatt die Programmsteuerung an eine andere Stelle der Prozedur weiterzugeben.

**On Error GoTo 0** deaktiviert die Fehlerbehandlung in der aktuellen Prozedur. Dabei wird die Programmzeile 0 nicht als Ausgangspunkt des Fehlerbehandlungscodes angegeben, auch wenn die Prozedur eine Programmzeile mit der Nummer 0 enthält. Ohne die Anweisung On Error GoTo 0 wird eine Fehlerbehandlungsroutine automatisch beim Verlassen einer Prozedur deaktiviert.

Um zu verhindern, daß ein Fehlerbehandlungscode ausgeführt wird, wenn kein Fehler aufgetreten ist, setzen Sie eine Exit Sub- oder Exit Function-Anweisung unmittelbar vor die Fehlerbehandlungsroutine, wie es das folgende Beispiel zeigt:

```
Sub Test()
    On Error GoTo Fehlerroutine
    ...
    Exit Sub
  Fehlerroutine:
    ...
    Resume Next
End Sub
```

### Beispiel 4: On Error-Fehlerbehandlung (B04)

Dieses Beispiel baut die Fehlerbehandlung mit der On Error-Anweisung in eine Prozedur ein. (Aus Platzgründen im Web).

```
Private Sub Form_Click ()
' Programm B04
Dim Drive, Msg   'Variablen deklarieren.
   On Error GoTo ErrorHandler
     ' Fehlerbehandlungroutine einrichten.
   Msg = "In dieser Demo soll eine
nichtexistierende Datei auf"
   Msg = Msg & "einem eventuell nicht
existierenden Laufwerk"
   Msg = Msg & "geöffnet werden. "
   Msg = Msg & "Im Fehlerfall zeigt die
Fehlerbehandlungsroutine eine "
   Msg = Msg & "entsprechende Meldung an."
   MsgBox Msg  'Erste Meldung anzeigen.
   ' Laufwerksbuchstaben zufällig bestimmen.
   Drive = Chr(Int((26) * Rnd + 1) + 64)
   Open Drive & ":\TEST\X.DAT" For Input As
#1    ' Datei öffnen.
   Close #1 ' Datei schließen.
Exit Sub
' Prozedur beenden, bevor Fehlerbehandlung
' beginnt.

ErrorHandler:
' Zeilenmarke für Fehlerbehandlungsroutine.

Select Case Err
  Case 53:
    Msg = "ERROR 53: Datei existiert nicht."
  Case 68:
    Msg = "ERROR 68: Laufwerk " & Drive & ":
nicht verfügbar."
  Case 76:
    Msg = "ERROR 76: Pfad existiert nicht."
  Case Else:
    Msg = "ERROR " & Err &  " ist
aufgetreten."
End Select

MsgBox Msg  ' Fehlermeldung anzeigen.
  Resume Next ' Prozedur fortsetzen.

End Sub
```

## Literaturverzeichnis

Es ist sehr schwierig, Empfehlungen aus der Vielzahl an vorhandener Literatur zu geben. Es sei an dieser Stelle nochmals auf die Wichtigkeit der **Online-Hilfe** und des **Lernprogramms** hingewiesen. Schließlich gilt, daß Programmieren nur durch Programmieren erlernt werden kann.

Dennoch können zum Beispiel folgende Werke empfohlen werden:

Peter MONADJEMI: Visual Basic 5 – Das Kompendium. Markt & Technik-Verlag, 1. Auflage, Haar/München 1997. ISBN 3-8272-5268-7.

Microsoft Visual Basic 5 – Schritt für Schritt. Microsoft Press-Verlag 1997. ISBN 3-86063-734-7

### Vorschau

In der nächsten Ausgabe wird dieser Visual-Basic-Lehrgang mit spezialisierten Kapiteln sowie mit einer Befehlsübersicht abgeschlossen.